

Más allá del testing

Sergio Gil
Christos Zisopoulos

Estrategias de testing

DDT: Development Driven Testing

DDT: Development Driven Testing

Escribir los tests después
de la primera iteración
del código

Por qué

Por qué

- A veces la especificación ~~no existe~~ es difusa, y hay poco tiempo

Por qué

- A veces la especificación ~~no existe~~ es difusa, y hay poco tiempo
- Es más fácil para empezar

Por qué

- A veces la especificación ~~no existe~~ es difusa, y hay poco tiempo
- Es más fácil para empezar
- Si tienes código ya escrito sin tests, es la única manera de que los tenga =;-)

Por qué

Por qué

- Si tu código (ya existente) no está escrito con TDD/BDD en mente, es más fácil el DDT

Por qué

- Si tu código (ya existente) no está escrito con TDD/BDD en mente, es más fácil el DDT
- Es **mucho** mejor que NO testear

Por qué

- Si tu código (ya existente) no está escrito con TDD/BDD en mente, es más fácil el DDT
- Es **mucho** mejor que NO testear
- Es lo que hace DHH =;-)

TDD: Test Driven Development

TDD: Test Driven Development

Escribir los tests
antes de implementar
el código

Por qué

Por qué

- Te fuerza a pensar en el código y en su diseño antes de escribirlo

Por qué

- Te fuerza a pensar en el código y en su diseño antes de escribirlo
- Te fuerza tener contratos de interacción concretos entre los componentes de tu código

Por qué

- Te fuerza a pensar en el código y en su diseño antes de escribirlo
- Te fuerza tener contratos de interacción concretos entre los componentes de tu código
- Escribir un método cuyos resultados ya conoces es más fácil que escribirlo y luego imaginar cuál será su salida

Por qué

- Te fuerza a pensar en el código y en su diseño antes de escribirlo
- Te fuerza tener contratos de interacción concretos entre los componentes de tu código
- Escribir un método cuyos resultados ya conoces es más fácil que escribirlo y luego imaginar cuál será su salida
- Si quieres refactorizar un método, te asegura que no vas a romper nada

BDD: Behaviour Driven Testing

BDD: Behaviour Driven Testing

Escribir, antes de implementar,
especificaciones que luego
sirvan para testear

Por qué

Por qué

- Es descriptivo

Por qué

- Es descriptivo
- Sirve de documentación (es legible por humanos normales)

Por qué

- Es descriptivo
- Sirve de documentación (es legible por humanos normales)
- Ayuda a crear mejores modelos y mejores interacciones entre ellos

Por qué

- Es descriptivo
- Sirve de documentación (es legible por humanos normales)
- Ayuda a crear mejores modelos y mejores interacciones entre ellos
- Es más fácil de escribir

Patrones y buenas prácticas

El buen código es
fácil de testear

El buen código es
fácil de testear

El código fácil de
testear es bueno

El buen código es
fácil de testear



El código fácil de
testear es bueno

Números *mágicos*

Números *mágicos*

- Hasta 7 acciones por controlador

Números *mágicos*

- Hasta 7 acciones por controlador
- Hasta 10 líneas por acción

Números *mágicos*

- Hasta 7 acciones por controlador
- Hasta 10 líneas por acción
- Si te hace falta más, sepáralo, te falta algo:
 - Un modelo
 - Un controlador
 - Una librería

Ley de Deméter

Ley de Deméter

- “Habla sólo con tus inmediatos amigos”

Ley de Deméter

- “Habla sólo con tus inmediatos amigos”
- Un objeto debería asumir lo mínimo posible acerca de la estructura o propiedades de nada más, incluyendo sus subcomponentes

```
post.comments.map(&:owner).map(&:email)  
post.comments.map(&:owner_email)  
post.commentator_emails
```

```
post.comments.map(&:owner).map(&:email)  
post.comments.map(&:owner_email)  
post.commentator_emails
```

¡Más fácil de *mockear*!

```
# Mal:
#
# post
# post.coments
# post.comments.map(&:owner)
# post.comments.map(&:owner).map(&:email)

before(:each) do

  @owner = User.new
  @owner.stubs(:email).returns('christos@the-cocktail.com')

  @comment = Comment.new
  @comment.stubs(:owner).returns(@owner)

  @post = Post.new
  @post.stubs(:comments).returns([@comment])
end

specify 'should return a list of comentator emails' do
  post.comments.map(&:owner).map(&:email).should.equal ['christos@the-cocktail.com']
end
```

```
# Bien:
#
# comment.owner_emails    -> owners.map(&:email)
# post.commentator_emails -> comments.map(&:owner_emails)
#

before(:each) do
  @post = Post.new
  @post.stubs(:commentator_emails).returns(['christos@the-  
cocktail.com'])
end

specify 'should return a list of comentator emails' do
  post.commentator_emails.should.equal ['christos@the-cocktail.com']
end
```

@@valid_model_attributes

@@valid_model_attributes

```
# en test_helper.rb
```

```
@@valid_post_attributes = { :title => 'Más allá del testing',  
                           :created_at => Time.now, :text => 'Bla, bla, bla...' }
```

@@valid_model_attributes

```
# en test_helper.rb
@@valid_post_attributes = { :title => 'Más allá del testing',
                             :created_at => Time.now, :text => 'Bla, bla, bla...' }

it 'should be invalid without title' do
  @post = Post.new(@@valid_post_attributes.except(:title))
  @post.should.not.validate
end
```

La decisión es QUÉ testear

La decisión es QUÉ testear

- No testes ActiveRecord

La decisión es QUÉ testear

- No testes ActiveRecord
- Testea las validaciones

La decisión es QUÉ testear

- No testes ActiveRecord
- Testea las validaciones
- Testea las interacciones entre modelos (teniendo en cuenta la Ley de Deméter)

La decisión es QUÉ testear

La decisión es QUÉ testear

- Testea siempre los *casos límite*

La decisión es QUÉ testear

- Testea siempre los *casos límite*
- Testea las vistas puntualmente

La decisión es QUÉ testear

- Testea siempre los *casos límite*
- Testea las vistas puntualmente
- Testea los controladores teniendo en cuenta la Ley de Deméter

La decisión es QUÉ testear

- Testea siempre los *casos límite*
- Testea las vistas puntualmente
- Testea los controladores teniendo en cuenta la Ley de Deméter
 - Un controlador debería manejar sólo su propio modelo

Algunas herramientas chulas

Rcov

Name	Total lines	Lines of code	Total coverage	Code coverage
TOTAL	616	409	78.6%	68.0%
app/controllers/application.rb	27	17	63.0%	47.1%
app/helpers/application_helper.rb	23	19	39.1%	26.3%
app/models/avatar.rb	31	12	100.0%	100.0%
app/models/tag.rb	31	17	100.0%	100.0%
app/models/tagging.rb	34	15	100.0%	100.0%
app/models/user.rb	115	72	98.3%	97.2%
lib/array.rb	6	6	100.0%	100.0%
lib/authenticated_system.rb	121	63	66.1%	34.9%
lib/authenticated_test_helper.rb	113	83	54.9%	38.6%
lib/date_time.rb	81	75	82.7%	81.3%
lib/string.rb	34	30	100.0%	100.0%

Generated using the [rcov code coverage analysis tool](#) for Ruby version 0.8.0.

Rcov

```
9
10 def create
11   errors = {}
12   @tags = Tag.parse_list(params[:tagging][:tag_names])
13   @taggings = @tags.map do |tag|
14     tagging = Tagging.new(:tag => tag, :author => current_user, :user => @user)
15     tagging.save
16     errors[tag.name] = tagging.errors
17     tagging
18   end
19   flash[:notice] = "¡OÃdo cocina! #{@user.public_name} es <em>#{@tags.map(&:name).map(&:desanitize).to_sentence}</em>."
20   flash[:errors] = errors.map do |tag_name, error_list|
21     error_list.full_messages.map do |message|
22       "<strong>#{tag_name}</strong>: #{message}"
23     end
24   end.flatten unless errors.empty?
25   redirect_to user_path(@user)
26 end
27
28 def index
29   @taggings = Tagging.find(
30     :all,
31     :order => 'created_at DESC',
32     :limit => 10
33   )
34 end
35
36 protected
37
38 def set_user
39   render_404 unless @user = User.find_by_login(params[:user_id])
40 end
41 end
```

Rcov

Rcov

- No es la panacea

Rcov

- No es la panacea
- Pero mola

Rcov

- No es la panacea
- Pero mola
- Uso:

Rcov

- No es la panacea
- Pero mola
- Uso:
 - Gema

Rcov

- No es la panacea
- Pero mola
- Uso:
 - Gema
 - Plugin

Rcov

- No es la panacea
- Pero mola
- Uso:
 - Gema
 - Plugin
 - Rake

Ojo: el test coverage despista a veces

Combínalo con cosas como
heckle (¡¡metateesting!!)

Integración continua

Integración continua

- Si tienes buenos tests, tardan un rato =;-)

Integración continua

- Si tienes buenos tests, tardan un rato =;-)
- Durante el desarrollo, sólo ejecutas los tests del código que tocas

Integración continua

- Si tienes buenos tests, tardan un rato =;-)
- Durante el desarrollo, sólo ejecutas los tests del código que tocas
- Para todo lo demás, CruiseControl.rb

Integración continua

CruiseControl.rb
Continuous Integration for Ruby

CruiseControlRB [Build Now](#) **rrussell** committed the checkin

build 568 (21 Nov) took 1 minute
567 (17 Nov)
566 (17 Nov)
565 (17 Nov)
563 (4 Nov) FAILED

Comments:
(RR) CCRB-160: 6 or more projects don't display correctly

RubyOnRails [Build Now](#) **bitsweat** committed the checkin

build **8189 (0:01) FAILED**
8188.1 (21 Nov) FAILED
8188 (21 Nov) incomplete
8186.1 (21 Nov) FAILED
8186 (21 Nov) FAILED

Comments:
Document that the cookie store is the default session store.
Susser, Jeremy Kemper]

¿Preguntas?

Muchas gracias

sergio.gil@the-cocktail.com

christos@the-cocktail.com

the-cocktail.com